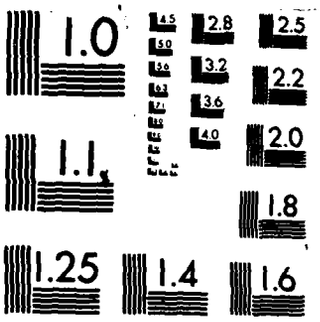
AD-A138 121 ANALYZING PROGRAM METHODOLOGIES USING SOFTWARE SCIENCE 1/1
(U) OHIO STATE UNIV RESEARCH FOUNDATION COLUMBUS
S H ZWEBEN JAN 84 ARO-17150.4-EL DAAG29-80-K-0061
UNCLASSIFIED F/G 9/2 NL

END
DATE FILMED
3-84
DTIC



UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: 17150.4-EL	2. GOVT ACCESSION NO.: N/A
3. RECIPIENT'S CATALOG NUMBER: N/A	
4. TITLE (and Subtitle): Analyzing Program Methodologies Using Software Science	5. TYPE OF REPORT & PERIOD COVERED: 1 Aug 80-31 Dec 83 Final Report
6. PERFORMING ORG. REPORT NUMBER: N/A	
7. AUTHOR(s): S. H. Zweben	8. CONTRACT OR GRANT NUMBER(s): DAAG29-80-K-0061
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Ohio State Univ, Columbus, Ohio 43212	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: N/A
11. CONTROLLING OFFICE NAME AND ADDRESS: U. S. Army Research Office, P. O. Box 12211, Research Triangle Park, NC 27709	12. REPORT DATE: Jan 84
	13. NUMBER OF PAGES: 10
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)	15. SECURITY CLASS. (of this report): Unclassified
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

THE VIEW, OPINIONS, AND/OR FINDINGS CONTAINED IN THIS REPORT ARE THOSE OF THE AUTHOR(S) AND SHOULD NOT BE CONSTRUED AS AN OFFICIAL DEPARTMENT OF THE ARMY POSITION, POLICY, OR DECISION, UNLESS SO DESIGNATED BY OTHER DOCUMENTATION.

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Computer Programs
Computer Programming

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

The ultimate goal of the research program is to enhance the quality of computer software. In order to accomplish this goal, however, there have to be agreed upon notions of just what quality means and how it can be assessed. This project sought to make contributions to our understanding of these issues. One of the specific objectives of this project was to study software science metrics in the COBOL arena, another objective concerned the evaluation of principles of software development. Research also sought to examine instruments alternative to the comprehension test which are easier to create but which are

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

20. ABSTRACT CONTINUED

still reliable and valid means of measuring one's understanding of a piece of software.

ARO 17150.4-E

RF Project 762340/713168
Final Report

ANALYZING PROGRAM METHODOLOGIES USING SOFTWARE SCIENCE

Stuart H. Zweben
Department of Computer and INformation Science

For the Period
August 1, 1980 - December 31, 1983

U.S. ARMY RESEARCH OFFICE
P.O. Box 12211
Research Triangle Park, N.C. 27709

Contract No. DAAG29-80-K-0061

January, 1984

The Ohio State University Research Foundation
1314 Kinnear Road
Columbus, Ohio 43212

Introduction

This is the final report of a research project which began in the summer of 1980 and continued through the end of 1983. This report will summarize the various objectives of the project, the approaches followed to accomplish these objectives, and the major results obtained. Appended to this report is a list of the participating personnel, and a list of the various scholarly activities associated with the project which were performed by these personnel.

Objectives

The ultimate goal of the research program is to enhance the quality of computer software. In order to accomplish this goal, however, there have to be agreed upon notions of just what quality means and how it can be assessed. This project sought to make contributions to our understanding of these issues.

In the past decade, it has become increasingly popular to measure features of the code itself, and to associate the resulting metrics with quality. One technique, due to Halstead, et al., has come to be known as "software science". The metrics of software science consist of functions of the operators and operands used in the code. Several researchers have studied these metrics, and many have found apparent relationships between the metrics and such behavioral characteristics as effort to write the code. One of the specific objectives of this project was to study the software science metrics in the COBOL arena, as the earlier work had not really been concerned with programs written in this language.

Another objective of this research concerned the evaluation of principles of software development. In many software engineering articles and reports, authors make statements concerning the DOs and DON'Ts for obtaining good software. These statements, however, are often based on no scientifically obtained evidence. Our research therefore was interested in seeing the extent to which some of these principles could be validated in a controlled laboratory setting.

The implementation of experiments to test hypotheses in software engineering is often hampered by the difficulty in measuring aspects of quality such as understandability. The standard means for assessing understandability, the comprehension test, is very time-consuming to create. We therefore sought to examine alternative instruments which are easier to create but which are still reliable and valid means of measuring one's understanding of a piece of software.

Approaches

In trying to apply the area of software science to COBOL, we first of all needed to create a software tool on which the various metrics of software science could be conveniently obtained. The next step was to analyze COBOL programs using the tool to see if the relationships from software science for which evidence existed in other programming languages could be shown to apply to COBOL. The goal here was to use a variety of programs, including those written by students in computer science courses as well as those written for production work by professional programmers. We were also aware of an effort by researchers at Purdue University in which COBOL programs were being analyzed using the software science metrics. We therefore desired to compare our results with those obtained by the Purdue group.

In deciding on our approach to evaluating principles of software development, we noted that several researchers already had begun to deal with coding principles. We therefore decided to focus our attention at the design phase of the software lifecycle, where several principles also exist. This approach also has the advantage of potentially allowing quality decisions to be reached earlier in the development process, when corrective action is typically less expensive.

We were interested in two types of investigations. One concerned the evaluation of several possible quantitative metrics which could be obtained from design documents, to see which appeared to provide the most information about the quality of the resulting software. The second concerned the evaluation of specific design principles, such as module coupling, in a controlled experimental setting. That is, we sought to determine if software

exhibiting features of good coupling (such as lack of global variable usage) could be shown superior to similar software whose coupling was poorer.

In attacking our final objective, that of improving the means of assessing understandability, we noted that researchers in psychology have often used a technique called the "cloze procedure" as an alternative to (multiple choice or short answer) comprehension questions. The cloze procedure is one which involves filling in missing blanks in a passage of text. The greater the subjects' ability to fill in the missing pieces, the greater the understanding of the material. Since cloze materials are very easy to construct, and have received some credibility in other domains, we decided to investigate this technique in the software domain.

Results

A tool for obtaining the software science metrics from COBOL programs was developed, tested, and enhanced during the project period. A technical report describing the tool's design and use was published and provided to the sponsor as part of a previous progress report. A large number of programs, from both student programming courses and production environments, were analyzed. A technical report detailing this investigation is in preparation, but the major conclusions are as follows.

The Halstead length relationship held up reasonably well for all classes of programs, but the components of the software that were included in the operator and operand counts strongly influenced this result. For instance, in smaller student programs, the length estimator was best when Data Division was included in the counting strategy. For larger programs, including production programs, counting or not counting Data Division didn't seem to make much difference. Overall, then, it appears to be appropriate to include Data Division in COBOL studies using software science metrics. In fact, this suggests that declarative components of programs in other languages might have strongly influenced earlier results in software science, and the data used in these early studies might well deserve re-examination along these lines to see whether the conculsions reached will change. If so, the reaction of the entire computer science community to the software science metrics might well

be different.

The language level estimator of software science could not be shown to be constant in any of our studies. We recommend that it is not useful as a metric at this time, especially when applied to an individual program.

The Halstead effort metric seemed to work very well for small programs, but was not very good for larger programs. We feel that the effort metric does not appropriately capture the effort required to integrate program components. Therefore, small programs which require little integration effort are relatively unaffected by this weakness, while larger programs can be greatly affected.

Other metrics, such as McCabe's cyclomatic number, Kafura's information flow metric, and Davis' chunk metric, were also studies in the "effort" domain. None of them provided consistently good results. We feel that there are to date no really valid metrics of software effort. The technical report discusses some approaches which may overcome the weaknesses of the metrics studied in this research.

The metrics obtained by our COBOL analyzer were compared with those obtained using a COBOL analyzer developed at Purdue University. The Purdue analyzer was implemented on Ohio State's Amdahl computer and the programs analyzed by the Ohio State COBOL analyzer were run on the Purdue analyzer. There were significant differences obtained in many of the operator and operand counts, due to differences in the counting strategies employed. The effects of these differences on the validity of the Halstead metrics, however, was not significant. That is, when one test program was compared to another using the OSU analyzer, the results were similar to those obtained when the same pair of programs were compared using the Purdue analyzer. It turned out that the same conclusions could be reached concerning the length estimator and the language level and effort estimators using either analyzer, for the set of programs studied. However, the fact that the absolute values of the metrics changed significantly cautions against reliably using the results obtained by different analyzers in comparisons of programs. This makes it extremely difficult to compare results obtained by different authors in the literature

on software metrics.

Our work in the evaluation of design methodology principles began with a study of several designs written according to the methodology of Structured Design espoused by Yourdon, et al, in a production environment. Several measurable properties of these designs were recorded, and their relationship to errors in development of the resulting system was analyzed. It turned out that the design features most closely related to development errors were those which had to do with the notion of coupling. The details of this study can be found in Doug Troy's masters thesis.

Next, we performed several experiments to determine if the principles which influence the level of coupling can be shown significant in a controlled environmental setting. We selected the feature of global variable usage for study because designs which differ along this single dimension are very far apart in the coupling hierarchy of Structured Design. Furthermore, materials for a controlled experiment which differed only along this dimension could easily be developed. The experiments tested this coupling feature against such attributes as understandability and modifiability. However, we could not produce any main effects for coupling level in these experiments. These results suggest that the effect of using global interface elements, rather than parameterized elements, is at best a second order influence on the quality of the software despite the prominent role which it plays in the coupling hierarchy of Structured Design.

Toward the end of the project period, we attempted to find other aspects of the design of software which might play a more significant role in the software's understandability. We tested two versions of a master file update program, one of which was developed according to a top-down design philosophy, and the other of which was developed using data abstraction design principles. The two versions were shown to differ in their understandability. In analyzing these differences, it seems to us that the principle of module cohesion is playing an important role. Future research will attempt to investigate this hypothesis more carefully.

Our investigations of the cloze procedure as a technique for assessing the

understandability of software brought some interesting results. An experimenter using a cloze technique has a great deal of freedom. For example, it is common to delete every nth word from prose texts. The choice of n does not seem to make much difference (as long as a minimun value of, say, at least 3 is selected), and n=5 is common. However, when applying the cloze procedure to the software domain, different results were obtained when n=3 and n=5 were used. This has a great bearing on the validity of the cloze procedure. We also noted that the kinds of "words" (things separated by blanks or punctuation) that were deleted made a significant difference in the performance of the subjects. We attempted to classify the nature of these deletions to explain these variations, and were fairly successful in this regard. The classification also predicted the results of our later experiments and those of another researcher who has studied the cloze procedure in the software domain. The forthcoming dissertation by Bill Hall will describe the details of these experiments and the classification scheme.

The work involving the cloze procedure has suggested several other kinds of experiments which can (and probably should) be done to determine the generality of our results. These include varying such factors as programming language, subject experience and problem domain. However, by far the more exciting outcome of this research has been its potential for more clearly identifying the characteristics of software that make it difficult to comprehend. As more is learned about classifying the "easy" and "hard" parts to complete in cloze tests, more useful code metrics (at least) may well result, and we may get closer to our objective of trying to quantify (and thereby possibly improve) at least one important dimension of software quality.

List of Personnel Who Worked on This Project

Stuart H. Zweben	Principal Investigator	Aug. 1980 - Dec. 1983
Douglas A. Troy	Graduate Student	Aug. 1980 - Mar. 1981 (not supported)
John B. Lonse	Graduate Research Assoc.	Oct. 1980 - Dec. 1983
Kin Chee Fung	Graduate Research Assoc.	Jan. 1981 - Sep. 1981
Eric Biefeld	Graduate Research Assoc.	Jul. 1981 - Aug. 1981
Narayan Debnath	Graduate Research Assoc.	Jul. 1982 - Jun. 1983
William E. Hall	Graduate Research Assoc.	Jul. 1982 - Dec. 1983
Julie Miller	Secretary	Sep. 1980 - Dec. 1982
Christine Poynter	Secretary	Apr. 1981 - Jun. 1981
Sheila Maginn	Secretary	Oct. 1981 - Mar. 1982
Amy Corneliussen	Secretary	Mar. 1982 - Jun. 1982

List of Scholarly Activities Associated with the Project

Publications

"Measuring the Quality of Structured Designs", Journal of Systems and Software, 2, 113-120 (1981); D.A. Troy and S.H. Zweben, co-authors.

"Evaluation of Design Methodologies, A Proposal", ACM SIGSOFT Software Engineering Notes, 7, 1, January, 1982, 60-69; H. Arisawa, G. Bergland, E. Bowers, J. Buxton, R. Kelley, N. Kerth, S. Saib, P. Ting, D. Troy, and S. Zweben, co-authors.

"Control Flow Complexity: Issues and Experience", Proceedings of the SHARE58 Conference, March 1982; S. Zweben, author.

"A Software Science Analyzer for COBOL", Technical Report 83-2, Computer and Information Science Research Center, Ohio State University, 1983; K.C. Fung, N. Debnath, and S. Zweben, co-authors.

"Experimental Evaluation of Software Design Principles: An Investigation into the Effect of Module Coupling on System Modifiability", Journal of Systems and Software, to appear; J. Lohse and S. Zweben, co-authors.

"A Study of the Application of Software Metrics to COBOL", Technical Report, Computer and Information Science Research Center, Ohio State University, 1984; in preparation; N. Debnath and S. Zweben, co-authors.

Presentations

"Software Science in Software Engineering", ACM Niagara Frontier Chapter, Buffalo, New York, March 1981 and ACM Rochester Chapter, Rochester, New York, March 1981; S. Zweben.

"The Status of Software Science in 1981", COMPSAC81, Chicago, Illinois, November 1981; S. Zweben.

"Software Metrics Research: An Academic Perspective", ACM Computer Science Conference, Indianapolis, Indiana, February 1982; S. Zweben.

"Control Flow Complexity: Issues and Experience", SHARE58 Conference, Los Angeles, California, March 1982; S. Zweben.

"Toward A Measure of System Coupling", Fifth Minnowbrook Workshop on Software, Blue Mountain Lake, New York, July 1982; J. Lohse and S. Zweben.

"Software Engineering: What Can It Do For You? What Can You Do For It?", ASIS82 Annual Conference, Columbus, Ohio, October 1982; S. Zweben.

"Software Metrics", Purdue University Computer Science Department's 20th Anniversary Symposium, West Lafayette, Indiana, November 1982; S. Zweben.

"Software Quality and the Halstead Metrics", ACM Greater Dayton Chapter, Dayton, Ohio, November 1982; S. Zweben.

"Principles of Software Engineering", Professional Development Seminar, The Ohio State University Office of Continuing Education, Columbus, Ohio, June 1983; S. Zweben.

"The Cloze Procedure and Program Complexity", Sixth Minnowbrook Workshop on Software, Blue Mountain Lake, New York, July 1983; W. Hall and S. Zweben.

Theses

"Measuring the Quality of Structured Designs", Masters Thesis, Department of Computer and Information Science, Ohio State University, March 1981; Douglas A. Troy, author; S.H. Zweben, advisor.

"The Cloze Procedure and Program Complexity", Ph.D. Thesis, Department of Computer and Information Science, Ohio State University, in preparation; William E. Hall, author; S.H. Zweben, advisor.


DAT
FILM